Package 'blockForest'

May 7, 2026

Type Package

Title Block Forests: Random Forests for Blocks of Clinical and Omics Covariate Data

Version 0.2.7

Date 2026-01-14

Description A random forest variant 'block forest' ('BlockForest') tailored to the prediction of binary, survival and continuous outcomes using block-structured covariate data, for example, clinical covariates plus measurements of a certain omics data type or multi-omics data, that is, data for which measurements of different types of omics data and/or clinical data for each patient exist. Examples of different omics data types include gene expression measurements, mutation data and copy number variation measurements.
Block forest are presented in Hornung & Wright (2019). The package includes four other random forest variants for multi-omics data: 'RandomBlock', 'BlockVarSel', 'VarProb', and 'SplitWeights'. These were also considered in Hornung & Wright (2019), but performed worse than block forest in their comparison study based on 20 real multi-omics data sets. Therefore, we recommend to use block forest ('BlockForest') in applications. The other random forest variants can, however, be consulted for academic purposes, for example, in the context of further methodological developments.
Reference: Hornung, R. & Wright, M. N. (2019) Block Forests: random forests for blocks of clinical and omics covariate data. BMC Bioinformatics 20:358. <doi:10.1186/s12859-019-2942-y>.

License GPL-3

Imports Rcpp (>= 0.11.2), Matrix, methods, survival

LinkingTo Rcpp, RcppEigen

Depends R (>= 3.1)

Suggests testthat

RoxygenNote 7.3.3

Encoding UTF-8

NeedsCompilation yes

URL https://github.com/bips-hb/blockForest, https://bips-hb.github.io/blockForest/

BugReports https://github.com/bips-hb/blockForest/issues

Author Marvin N. Wright [aut, cre],
Roman Hornung [aut]

Maintainer Marvin N. Wright <cran@wrig.de>

Repository CRAN

Date/Publication 2026-01-14 09:20:02 UTC

Contents

`blockfor` *Random Forest variants for block-structured covariate data*

Description

Implements five Random Forest variants for prediction of binary, survival and metric outcomes using block-structured covariate data, for example, clinical covariates plus measurements of a certain omics data type or multi-omics data, that is, data for which measurements of different types of omics data and/or clinical data for each patient exist. For example, for the task of predicting survival for each patient there might be available clinical covariates, gene expression measurements, mutation data, and copy number variation measurements.
The group of covariates corresponding to one specific data type is denoted as a 'block'.
NOTE: We strongly recommend using the variant "BlockForest" (or "block forest") in applications. The other four variants performed worse than "BlockForest" in the analysis by Hornung & Wright (2019). Using 20 real multi-omics data sets Hornung & Wright (2019) compared all five variants with each other and with alternatives, in particular with Random Survival Forest as existing reference method. The ranking of the performances of the five variants was as follows in the comparison study by Hornung & Wright (2019): 1) "BlockForest", 2) "RandomBlock", 3) "BlockVarSel", 4) "VarProb", 5) "SplitWeights".
Each of the five variants uses a different split selection algorithm. For details, see Hornung & Wright (2019).
Note that this R package is a fork of the R package ranger.
NOTE ALSO: Including the clinical block mandatorily in the split point selection can considerably improve the prediction performance. Whether or not this is the case, depends on the level of predictive information contained in the clinical block. We recommend trying out including the clinical

block mandatorily to see, whether this improves prediction performance in the particular application. Note that in the case of including the clinical block mandatorily and having more than only one omics block, "RandomBlock" performed (slightly) better than "BlockForest" in the comparison study by Hornung & Wright (2019). Including the clinical block mandatorily can be performed by setting the function argument 'always.select.block' of 'blockfor()' to the index of the clinical block (e.g., if the clinical block would be the second block in order, we would set always.select.block=2).

Usage

```
blockfor(
  X,
  y,
  blocks,
  block.method = "BlockForest",
  num.trees = 2000,
  mtry = NULL,
  nsets = 300,
  num.trees.pre = 1500,
  splitrule = "extratrees",
  always.select.block = 0,
  ...
)
```

Arguments

X	Covariate matrix. observations in rows, variables in columns.
y	Target variable. If the outcome is binary, this is a factor with two levels. If the outcome is metric, this is a numeric vector. If the outcome is a survival outcome, this is a matrix with two columns, where the first column contains the vector of survival/censoring times (one for each observation) and the second column contains the status variable, that has the value '1' if the corresponding time is a survival time and '0' if that time is a censoring time.
blocks	A list of length equal to the number M of blocks considered. Each entry contains the vector of column indices in 'X' of the covariates in one of the M blocks.
block.method	Forest variant to use. One of the following: "BlockForest" (default), "RandomBlock", "BlockVarSel", "VarProb", "SplitWeights". The latter listing is ordered according to the performances of these variants in the comparison study by Hornung & Wright (2019), with the best variant being listed first.
num.trees	Number of trees in the forest.
mtry	This is either a number specifying the number of variables sampled for each split from all variables (for variants "VarProb" and "SplitWeights") or a vector of length equal to the number of blocks, where the m-th entry of the vector gives the number of variables to sample from block m (for variants "BlockForest", "RandomBlock", and "BlockVarSel"). The default values are sqrt(p_1) + sqrt(p_2) + ... sqrt(p_M) and (sqrt(p_1), sqrt(p_2), ..., sqrt(p_M)), respectively, where p_m denotes the number of variables in the m-th block (m = 1, ..., M) and sqrt() denoted the square root function.

`nsets` Number of sets of tuning parameter values generated randomly in the optimization of the tuning parameters. Each variant has a tuning parameter for each block, that is, there are M tuning parameters for each variant. These tuning parameters are optimized in the following way: 1. Generate random sets of tuning parameter values and measure there adequateness: For j = 1,..., nsets: a) Generate a random set of tuning parameter values; b) Construct a forest (with num.trees.pre trees) using the set of tuning parameter values generated in a); c) Record the out-of-bag (OOB) estimated prediction error of the forest constructed in b); 2. Use the set of tuning parameter values generated in 1. that is associated with the smallest OOB estimated prediction error.

`num.trees.pre` Number of trees in each forest constructed during the optimization of the tuning parameter values, see 'nsets' for details.

`splitrule` Splitting rule. Default "extratrees" (for computational efficiency). For other options see `blockForest`.

`always.select.block` Number of block to make always available for splitting (e.g. clinical covariates).

`...` Parameters passed to `blockForest`, such as `num.threads`, etc. See `blockForest` for details.

Value

`blockfor` returns a list containing the following components:

`forest` object of class `"blockForest"`. Constructed forest.

`paramvalues` vector of length M. Optimized tuning parameter value for each block.

`biased_oob_error_donotuse` numeric. OOB estimated prediction error. NOTE: This estimate should not be used, because it is (highly) optimistic (i.e, too small), because the data set was used twice - for optimizing the tuning parameter values and for estimating the prediction error. Instead, cross-validation should be used to estimate the prediction error.

Author(s)

Roman Hornung, Marvin N. Wright

References

- Hornung, R. & Wright, M. N. (2019) Block Forests: random forests for blocks of clinical and omics covariate data. BMC Bioinformatics 20:358. doi:10.1186/s128590192942y.
- Breiman, L. (2001). Random forests. Mach Learn, 45(1), 5-32. doi:10.1023/A:1010933404324.
- Wright, M. N. & Ziegler, A. (2017). ranger: A Fast Implementation of Random Forests for High Dimensional Data in C++ and R. J Stat Softw 77:1-17. doi:10.18637/jss.v077.i01.

Examples

```
# NOTE: There is no association between covariates and response for the
# simulated data below.
# Moreover, the input parameters of blockfor() are highly unrealistic
# (e.g., nsets = 10 is specified much too small).
# The purpose of the shown examples is merely to illustrate the
# application of blockfor().


# Generate data:
################

set.seed(1234)

# Covariate matrix:
X <- cbind(matrix(nrow=40, ncol=5, data=rnorm(40*5)),
           matrix(nrow=40, ncol=30, data=rnorm(40*30, mean=1, sd=2)),
           matrix(nrow=40, ncol=100, data=rnorm(40*100, mean=2, sd=3)))

# Block variable (list):
blocks <- rep(1:3, times=c(5, 30, 100))
blocks <- lapply(1:3, function(x) which(blocks==x))

# Binary outcome:
ybin <- factor(sample(c(0,1), size=40, replace=TRUE), levels=c(0,1))

# Survival outcome:
ysurv <- cbind(rnorm(40), sample(c(0,1), size=40, replace=TRUE))

# Application to binary outcome:
################################

blockforobj <- blockfor(X, ybin, num.trees = 100, replace = TRUE, blocks=blocks,
                        nsets = 10, num.trees.pre = 50, splitrule="extratrees",
                        block.method = "BlockForest")
# Tuning parameter estimates (see Hornung & Wright (2019)):
blockforobj$paramvalues

# Application to survival outcome:
##################################

blockforobj <- blockfor(X, ysurv, num.trees = 100, replace = TRUE, blocks=blocks,
                        nsets = 10, num.trees.pre = 50, splitrule="extratrees",
                        block.method = "BlockForest")
blockforobj$paramvalues
```

Description

Block forests without parameter tuning. Use blockfor for standard interface. This function is called by blockfor and will rarely be considered directly by the user (since parameter tuning is required in applications).

Usage

```
blockForest(
  formula = NULL,
  data = NULL,
  num.trees = 500,
  mtry = NULL,
  importance = "none",
  write.forest = TRUE,
  probability = FALSE,
  min.node.size = NULL,
  replace = TRUE,
  sample.fraction = ifelse(replace, 1, 0.632),
  case.weights = NULL,
  splitrule = NULL,
  num.random.splits = 1,
  alpha = 0.5,
  minprop = 0.1,
  split.select.weights = NULL,
  always.split.variables = NULL,
  blocks = NULL,
  block.method = "BlockForest",
  block.weights = NULL,
  respect.unordered.factors = NULL,
  scale.permutation.importance = FALSE,
  keep.inbag = FALSE,
  holdout = FALSE,
  quantreg = FALSE,
  num.threads = NULL,
  save.memory = FALSE,
  verbose = TRUE,
  seed = NULL,
  dependent.variable.name = NULL,
  status.variable.name = NULL,
  classification = NULL
)
```

Arguments

formula	Object of class formula or character describing the model to fit. Interaction terms supported only for numerical variables.
data	Training data of class data.frame, matrix, dgCMatrix (Matrix) or gwaa.data (GenABEL).

`num.trees` Number of trees.

`mtry` This is either a number specifying the number of variables sampled for each split from all variables (for variants "VarProb" and "SplitWeights") or a vector of length equal to the number of blocks, where the m-th entry of the vector gives the number of variables to sample from block m (for variants "BlockForest", "RandomBlock", and "BlockVarSel"). The default values are sqrt(p_1) + sqrt(p_2) + ... sqrt(p_M) and (sqrt(p_1), sqrt(p_2), ..., sqrt(p_M)), respectively, where p_m denotes the number of variables in the m-th block (m = 1, ..., M) and sqrt() denoted the square root function.

`importance` Variable importance mode, one of 'none', 'impurity', 'impurity_corrected', 'permutation'. The 'impurity' measure is the Gini index for classification, the variance of the responses for regression and the sum of test statistics (see `splitrule`) for survival.

`write.forest` Save `blockForest.forest` object, required for prediction. Set to `FALSE` to reduce memory usage if no prediction intended.

`probability` Grow a probability forest as in Malley et al. (2012).

`min.node.size` Minimal node size. Default 1 for classification, 5 for regression, 3 for survival, and 10 for probability.

`replace` Sample with replacement.

`sample.fraction`
Fraction of observations to sample. Default is 1 for sampling with replacement and 0.632 for sampling without replacement. For classification, this can be a vector of class-specific values.

`case.weights` Weights for sampling of training observations. Observations with larger weights will be selected with higher probability in the bootstrap (or subsampled) samples for the trees.

`splitrule` Splitting rule, default "extratrees". Other options are "gini" for classification and probability estimation, "variance", or "maxstat" for regression and "logrank", "C" or "maxstat" for survival.

`num.random.splits`
For "extratrees" splitrule.: Number of random splits to consider for each candidate splitting variable.

`alpha` For "maxstat" splitrule: Significance threshold to allow splitting.

`minprop` For "maxstat" splitrule: Lower quantile of covariate distribution to be considered for splitting.

`split.select.weights`
Numeric vector with weights between 0 and 1, representing the probability to select variables for splitting. Alternatively, a list of size num.trees, containing split select weight vectors for each tree can be used. Use this for the "VarProb" variant.

`always.split.variables`
Character vector with variable names to be always selected in addition to the `mtry` variables tried for splitting.

`blocks` Block memberships of the variables. See `blockfor` for details.

`block.method` Variant to use. Options are: "BlockForest" (default), "RandomBlock", "BlockVarSel", "SplitWeights".

`block.weights` Tuning parameter values for the blocks in the variants. A vector of length equal to the number of blocks or a list with vectors containing tree-wise values. For block.method='RandomBlock' these are the block sample probabilities.

`respect.unordered.factors`
Handling of unordered factor covariates. One of 'ignore', 'order' and 'partition'. For the "extratrees" splitrule the default is "partition" for all other splitrules 'ignore'. Alternatively TRUE (='order') or FALSE ='ignore') can be used. See below for details.

`scale.permutation.importance`
Scale permutation importance by standard error as in (Breiman 2001). Only applicable if permutation variable importance mode selected.

`keep.inbag` Save how often observations are in-bag in each tree.

`holdout` Hold-out mode. Hold-out all samples with case weight 0 and use these for variable importance and prediction error.

`quantreg` Prepare quantile prediction as in quantile regression forests (Meinshausen 2006). Regression only. Set `keep.inbag = TRUE` to prepare out-of-bag quantile prediction.

`num.threads` Number of threads. Default is number of CPUs available.

`save.memory` Use memory saving (but slower) splitting mode. No effect for survival and GWAS data. Warning: This option slows down the tree growing, use only if you encounter memory problems.

`verbose` Show computation status and estimated runtime.

`seed` Random seed. Default is `NULL`, which generates the seed from `R`. Set to `0` to ignore the `R` seed.

`dependent.variable.name`
Name of dependent variable, needed if no formula given. For survival forests this is the time variable.

`status.variable.name`
Name of status variable, only applicable to survival data and needed if no formula given. Use 1 for event and 0 for censoring.

`classification` Only needed if data is a matrix. Set to `TRUE` to grow a classification forest.

Details

By default, blockForest uses all available threads. The default can be changed with: (1) `num.threads` in blockForest/blockfor/predict call, (2) environment variable R_BLOCKFOREST_NUM_THREADS, (3) `options(blockforest.num.threads = N)`, (4) `options(Ncpus = N)`, with precedence in that order.

See `blockfor` and the `ranger` package.

Value

Object of class `blockForest` with elements

`forest` Saved forest (If write.forest set to TRUE). Note that the variable IDs in the `split.varIDs` object do not necessarily represent the column number in R.

`predictions` Predicted classes/values, based on out of bag samples (classification and regression only).

`variable.importance`
Variable importance for each independent variable.

`prediction.error`
Overall out of bag prediction error. For classification this is the fraction of missclassified samples, for probability estimation and regression the mean squared error and for survival one minus Harrell's C-index.

`r.squared` R squared. Also called explained variance or coefficient of determination (regression only). Computed on out of bag data.

`confusion.matrix`
Contingency table for classes and predictions based on out of bag samples (classification only).

`unique.death.times`
Unique death times (survival only).

`chf` Estimated cumulative hazard function for each sample (survival only).

`survival` Estimated survival function for each sample (survival only).

`call` Function call.

`num.trees` Number of trees.

`num.independent.variables`
Number of independent variables.

`mtry` Value of mtry used.

`min.node.size` Value of minimal node size used.

`treetype` Type of forest/tree. classification, regression or survival.

`importance.mode`
Importance mode used.

`num.samples` Number of samples.

`inbag.counts` Number of times the observations are in-bag in the trees.

Author(s)

Marvin N. Wright

References

- Hornung, R. & Wright, M. N. (2019) Block Forests: random forests for blocks of clinical and omics covariate data. BMC Bioinformatics 20:358. doi:10.1186/s128590192942y.
- Wright, M. N. & Ziegler, A. (2017). ranger: A Fast Implementation of Random Forests for High Dimensional Data in C++ and R. J Stat Softw 77:1-17. doi:10.18637/jss.v077.i01.
- Schmid, M., Wright, M. N. & Ziegler, A. (2016). On the use of Harrell's C for clinical risk prediction via random survival forests. Expert Syst Appl 63:450-459. doi:10.1016/j.eswa.2016.07.018.

- Wright, M. N., Dankowski, T. & Ziegler, A. (2017). Unbiased split variable selection for random survival forests using maximally selected rank statistics. Stat Med. doi:10.1002/sim.7212.
- Breiman, L. (2001). Random forests. Mach Learn, 45(1), 5-32. doi:10.1023/A:1010933404324.
- Ishwaran, H., Kogalur, U. B., Blackstone, E. H., & Lauer, M. S. (2008). Random survival forests. Ann Appl Stat 2:841-860. doi:10.1097/JTO.0b013e318233d835.
- Malley, J. D., Kruppa, J., Dasgupta, A., Malley, K. G., & Ziegler, A. (2012). Probability machines: consistent probability estimation using nonparametric learning machines. Methods Inf Med 51:74-81. doi:10.3414/ME00010052.
- Hastie, T., Tibshirani, R., Friedman, J. (2009). The Elements of Statistical Learning. Springer, New York. 2nd edition.
- Geurts, P., Ernst, D., Wehenkel, L. (2006). Extremely randomized trees. Mach Learn 63:3-42. doi:10.1007/s10994006622 61.
- Meinshausen (2006). Quantile Regression Forests. J Mach Learn Res 7:983-999. https://www.jmlr.org/papers/v7/meinshausen06a.html.

See Also

predict.blockForest

Examples

```
require(blockForest)

# Standard Block Forest
blockForest(Species ~ ., iris,
            blocks = list(1:2, 3:4),
            mtry = c(1, 2),
            block.weights = c(0.1, 0.9),
            block.method = "BlockForest")

# Without blocks, grow standard random forest
blockForest(Species ~ ., iris)
```

`predict.blockForest` *Prediction using Random Forest variants for block-structured covariate data*

Description

This function is to be applied to the entry 'forest' of the output of `blockfor`. See the example section for illustration.

Usage

```
## S3 method for class 'blockForest'
predict(
  object,
  data = NULL,
  predict.all = FALSE,
  num.trees = object$num.trees,
  type = "response",
  se.method = "infjack",
  quantiles = c(0.1, 0.5, 0.9),
  seed = NULL,
  num.threads = NULL,
  verbose = TRUE,
  ...
)
```

Arguments

object	blockForest object.
data	New test data of class data.frame or gwaa.data (GenABEL).
predict.all	Return individual predictions for each tree instead of aggregated predictions for all trees. Return a matrix (sample x tree) for classification and regression, a 3d array for probability estimation (sample x class x tree) and survival (sample x time x tree).
num.trees	Number of trees used for prediction. The first num.trees in the forest are used.
type	Type of prediction. One of 'response', 'se', 'terminalNodes', 'quantiles' with default 'response'. See below for details.
se.method	Method to compute standard errors. One of 'jack', 'infjack' with default 'infjack'. Only applicable if type = 'se'. See below for details.
quantiles	Vector of quantiles for quantile prediction. Set type = 'quantiles' to use.
seed	Random seed. Default is NULL, which generates the seed from R. Set to 0 to ignore the R seed. The seed is used in case of ties in classification mode.
num.threads	Number of threads. Default is number of CPUs available.
verbose	Verbose output on or off.
...	further arguments passed to or from other methods.

Details

For type = 'response' (the default), the predicted classes (classification), predicted numeric values (regression), predicted probabilities (probability estimation) or survival probabilities (survival) are returned. For type = 'se', the standard error of the predictions are returned (regression only). The jackknife-after-bootstrap or infinitesimal jackknife for bagging is used to estimate the standard errors based on out-of-bag predictions. See Wager et al. (2014) for details. For type = 'terminalNodes', the IDs of the terminal node in each tree for each observation in the given dataset are returned. For type = 'quantiles', the selected quantiles for each observation are estimated. See Meinshausen (2006) for details.

If type = 'se' is selected, the method to estimate the variances can be chosen with se.method. Set se.method = 'jack' for jackknife-after-bootstrap and se.method = 'infjack' for the infinitesimal jackknife for bagging.

For classification and predict.all = TRUE, a factor levels are returned as numerics. To retrieve the corresponding factor levels, use rf$forest$levels, if rf is the ranger object.

Value

Object of class blockForest.prediction with elements

predictions	Predicted classes/values (only for classification and regression)
unique.death.times	Unique death times (only for survival).
chf	Estimated cumulative hazard function for each sample (only for survival).
survival	Estimated survival function for each sample (only for survival).
num.trees	Number of trees.
num.independent.variables	Number of independent variables.
treetype	Type of forest/tree. Classification, regression or survival.
num.samples	Number of samples.

Author(s)

Marvin N. Wright

References

- Wright, M. N. & Ziegler, A. (2017). ranger: A Fast Implementation of Random Forests for High Dimensional Data in C++ and R. J Stat Softw 77:1-17. doi:10.18637/jss.v077.i01.
- Wager, S., Hastie T., & Efron, B. (2014). Confidence Intervals for Random Forests: The Jackknife and the Infinitesimal Jackknife. J Mach Learn Res 15:1625-1651. https://jmlr.org/papers/v15/wager14a.html.
- Meinshausen (2006). Quantile Regression Forests. J Mach Learn Res 7:983-999. https://www.jmlr.org/papers/v7/meinshausen06a.html.

See Also

blockForest

Examples

```
# NOTE: There is no association between covariates and response for the
# simulated data below.
# Moreover, the input parameters of blockfor() are highly unrealistic
# (e.g., nsets = 10 is specified much too small).
# The purpose of the shown examples is merely to illustrate the
# application of predict.blockForest().


# Generate data:
#################
```

```
set.seed(1234)

# Covariate matrix:
X <- cbind(matrix(nrow=40, ncol=5, data=rnorm(40*5)),
           matrix(nrow=40, ncol=30, data=rnorm(40*30, mean=1, sd=2)),
           matrix(nrow=40, ncol=100, data=rnorm(40*100, mean=2, sd=3)))
colnames(X) <- paste("X", 1:ncol(X), sep="")

# Block variable (list):
block <- rep(1:3, times=c(5, 30, 100))
block <- lapply(1:3, function(x) which(block==x))

# Binary outcome:
ybin <- factor(sample(c(0,1), size=40, replace=TRUE), levels=c(0,1))

# Survival outcome:
ysurv <- cbind(rnorm(40), sample(c(0,1), size=40, replace=TRUE))



# Divide in training and test data:

Xtrain <- X[1:30,]
Xtest <- X[31:40,]

ybintrain <- ybin[1:30]
ybintest <- ybin[31:40]

ysurvtrain <- ysurv[1:30,]
ysurvtest <- ysurv[31:40,]




# Binary outcome: Apply algorithm to training data and obtain predictions
# for the test data:
##############################################################################

# Apply a variant to the training data:

blockforobj <- blockfor(Xtrain, ybintrain, num.trees = 100, replace = TRUE, block=block,
                        nsets = 10, num.trees.pre = 50, splitrule="extratrees",
                        block.method = "SplitWeights")
blockforobj$paramvalues


# Obtain prediction for the test data:

(predres <- predict(blockforobj$forest, data = Xtest, block.method = "SplitWeights"))
predres$predictions
```

```
# Survival outcome: Apply algorithm to training data and obtain predictions
# for the test data:
##############################################################################

# Apply a variant to the training data:

blockforobj <- blockfor(Xtrain, ysurvtrain, num.trees = 100, replace = TRUE, block=block,
                        nsets = 10, num.trees.pre = 50, splitrule="extratrees",
                        block.method = "SplitWeights")
blockforobj$paramvalues


# Obtain prediction for the test data:

(predres <- predict(blockforobj$forest, data = Xtest, block.method = "SplitWeights"))
rowSums(predres$chf)
```

`predictions.blockForest` *blockForest predictions*

Description

Extract training data predictions of blockForest object.

Usage

```
## S3 method for class 'blockForest'
predictions(x, ...)
```

Arguments

`x`	blockForest object.
`...`	Further arguments passed to or from other methods.

Value

Predictions: Classes for Classification forests, Numerical values for Regressions forests and the estimated survival functions for all individuals for Survival forests.

Author(s)

Marvin N. Wright

See Also

`blockForest`

predictions.blockForest.prediction

blockForest predictions

Description

Extract predictions of blockForest prediction object.

Usage

```
## S3 method for class 'blockForest.prediction'
predictions(x, ...)
```

Arguments

x	blockForest prediction object.
...	Further arguments passed to or from other methods.

Value

Predictions: Classes for Classification forests, Numerical values for Regressions forests and the estimated survival functions for all individuals for Survival forests.

Author(s)

Marvin N. Wright

See Also

blockForest

timepoints.blockForest

blockForest timepoints

Description

Extract unique death times of blockForest Survival forest

Usage

```
## S3 method for class 'blockForest'
timepoints(x, ...)
```

Arguments

x	blockForest Survival forest object.
...	Further arguments passed to or from other methods.

Value

Unique death times

Author(s)

Marvin N. Wright

See Also

blockForest

---

timepoints.blockForest.prediction
blockForest timepoints

Description

Extract unique death times of blockForest Survival prediction object.

Usage

```
## S3 method for class 'blockForest.prediction'
timepoints(x, ...)
```

Arguments

x	blockForest Survival prediction object.
...	Further arguments passed to or from other methods.

Value

Unique death times

Author(s)

Marvin N. Wright

See Also

blockForest

`treeInfo` *Tree information in human readable format*

Description

Extract tree information of a `blockForest` object.

Usage

```
treeInfo(object, tree = 1)
```

Arguments

`object`	`blockForest` object.
`tree`	Number of the tree of interest.

Details

Node and variable ID's are 0-indexed, i.e., node 0 is the root node. If the formula interface is used in the `blockForest` call, the variable ID's are usually different to the original data used to grow the tree. Refer to the variable name instead to be sure.

Splitting at unordered factors (nominal variables) depends on the option `respect.unordered.factors` in the `blockForest` call. For the "ignore" and "order" approaches, all values smaller or equal the `splitval` value go to the left and all values larger go to the right, as usual. However, with "order" the values correspond to the order in `object$forest$covariate.levels` instead of the original order (usually alphabetical). In the "partition" mode, the `splitval` values for unordered factor are comma separated lists of values, representing the factor levels (in the original order) going to the left.

Value

A data.frame with the columns

`nodeID`	The nodeID, 0-indexed.
`leftChild`	ID of the left child node, 0-indexed.
`rightChild`	ID of the right child node, 0-indexed.
`splitvarID`	ID of the splitting variable, 0-indexed. Caution, the variable order changes if the formula interface is used.
`splitvarName`	Name of the splitting variable.
`splitval`	The splitting value. For numeric or ordinal variables, all values smaller or equal go to the left, larger values t
`terminal`	Logical, TRUE for terminal nodes.
`prediction`	One column with the predicted class (factor) for classification and the predicted numerical value for regressi

Author(s)

Marvin N. Wright

Examples

```
require(blockForest)
rf <- blockForest(Species ~ ., data = iris)
treeInfo(rf, 1)
```

Index